Original document

PERIPHERAL EQUIPMENT CONTROL SYSTEM

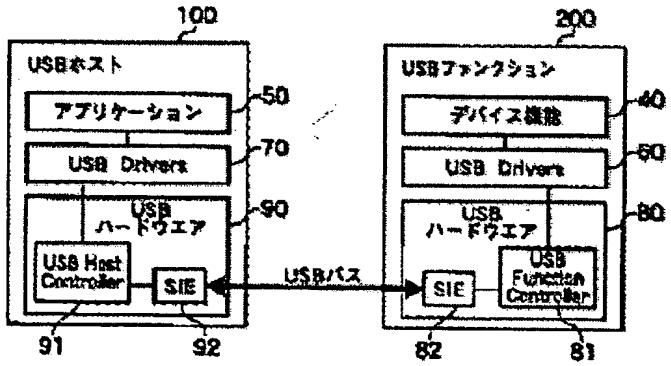
Publication number: JP2001222503
Publication date: 2001-08-17
Inventor: SENDA SHIGEYA
Applicant: RICOH KK
Classification:
- international: ***G06F13/14; B41J29/38; G06F3/12;*** **G06F13/14; B41J29/38; G06F3/12;** (IPC1-7): G06F13/14; B41J29/38; G06F3/12
- European:
Application number: JP20000030800 20000208
Priority number(s): JP20000030800 20000208

View INPADOC patent family
View list of citing documents

Report a data error here

Abstract of **JP2001222503**

PROBLEM TO BE SOLVED: To provide a peripheral equipment control system, with which the communication path of USB for plural devices can be secured without requiring a large number of end points and complicating a device driver and the existence of a port for debugging maintenance can not be known.
SOLUTION: A descriptor switching demand vender request is issued from a device driver 70 to a function 200, when switching is enabled, a device function 40 to be utilized, for example, is switched from print application to a debugging maintenance application and ACK is returned to a host 100 on a status stage. Afterwards, by performing connect signal disconnecting processing and pull-up recovering processing, on the host side, which interprets a new device, the descriptor acquiring operation of the device is performed and a class driver according to the acquired switched device is made active.



Data supplied from the *esp@cenet* database - Worldwide

Original document

PERIPHERAL EQUIPMENT CONTROL SYSTEM

Publication number: JP2001222503
Publication date: 2001-08-17

Inventor: SENDA SHIGEYA
Applicant: RICOH KK

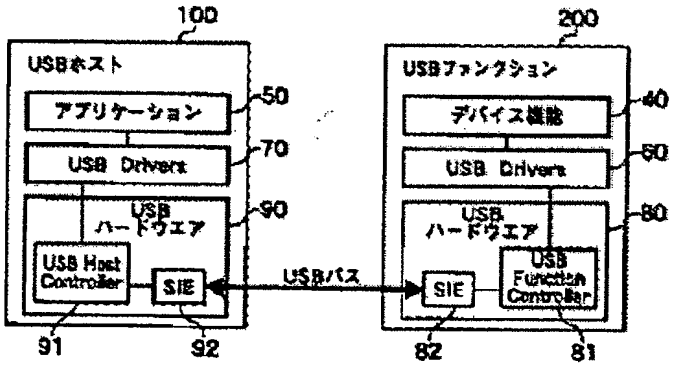
Classification:
- international: ***G06F13/14; B41J29/38; G06F3/12;*** **G06F13/14; B41J29/38; G06F3/12;** (IPC1-7): G06F13/14; B41J29/38; G06F3/12
- European:
Application number: JP20000030800 20000208
Priority number(s): JP20000030800 20000208

View INPADOC patent family
View list of citing documents

Report a data error here

Abstract of **JP2001222503**

PROBLEM TO BE SOLVED: To provide a peripheral equipment control system, with which the communication path of USB for plural devices can be secured without requiring a large number of end points and complicating a device driver and the existence of a port for debugging maintenance can not be known.
SOLUTION: A descriptor switching demand vender request is issued from a device driver 70 to a function 200, when switching is enabled, a device function 40 to be utilized, for example, is switched from print application to a debugging maintenance application and ACK is returned to a host 100 on a status stage. Afterwards, by performing connect signal disconnecting processing and pull-up recovering processing, on the host side, which interprets a new device, the descriptor acquiring operation of the device is performed and a class driver according to the acquired switched device is made active.



(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2001－222503
(P2001－222503A)
(43)公開日　平成13年８月17日(2001.8.17)

(51)Int.Cl.'	識別記号	ＦＩ		テーマコード'(参考)
Ｇ０６Ｆ　13/14	３３０	Ｇ０６Ｆ　13/14	３３０Ａ	２Ｃ０６１
Ｂ４１Ｊ　29/38		Ｂ４１Ｊ　29/38	Ｚ	５Ｂ０１４
Ｇ０６Ｆ　3/12		Ｇ０６Ｆ　3/12	Ａ	５Ｂ０２１

審査請求　未請求　請求項の数８　ＯＬ　（全 12 頁）

(21)出願番号　特願2000－30800(P2000－30800)

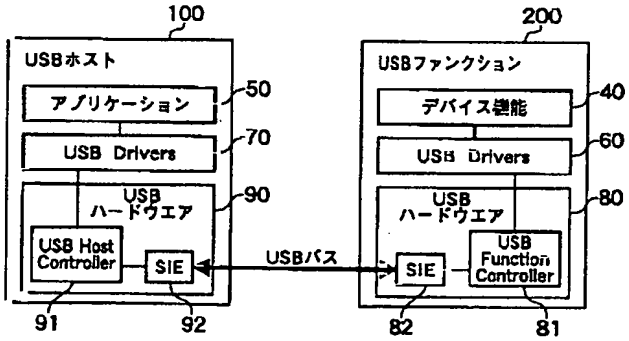
(22)出願日　平成12年２月８日(2000.2.8)

(71)出願人　000006747
株式会社リコー
東京都大田区中馬込１丁目３番６号
(72)発明者　千田　滋也
東京都大田区中馬込１丁目３番６号株式会社リコー内
(74)代理人　100110319
弁理士　根本　恵司

Ｆターム(参考)　2C061 AP03 AP04 AP07 HQ01 HQ20
5B014 EB03 GD05 GD22 GD47 GE05
HA02 HA07 HC04 HC08
5B021 AA01 BB01 CC05

(54)【発明の名称】　周辺機器制御システム

(57)【要約】
【課題】　多くのエンドポイントを必要とせず、デバイスドライバを複雑化せずに複数のデバイス用のUSBの通信路が確保でき、又デバック・メンテナンス用ポート等の存在を知られずにすむ周辺機器制御システムを提供する。
【解決手段】　デバイスドライバ７０からディスクリプタ切替え要求ベンダリクエストをファンクション２００へ発行し、切替え可能であれば、利用するデバイス機能４０例えばプリントアプリからデバック・メンテナンスアプリへ切替え、ステータスステージでACKをホスト１００に返送する。この後、コネクト信号断処理、プルアップの復活処理をし、これにより新しいデバイスが接続されたと解釈するホスト側ではデバイスのディスクリプタ取得動作を行い、取得した切替後のデバイスに従ったクラスドライバがアクティブにされる。



【特許請求の範囲】
【請求項１】　ホストと周辺機器をUSB方式により接続する周辺機器制御システムにおいて、ホストがコンフィギュレーション切替えベンダリクエストを発行する機能を備えたデバイスドライバを有することを特徴とする周辺機器制御システム。
【請求項２】　請求項１に記載された周辺機器制御システムにおいて、前記ホストは、デバイスドライバをコンフィギュレーション毎に有することを特徴とする周辺機器制御システム。
【請求項３】　請求項１又は２に記載された周辺機器制御システムにおいて、前記コンフィギュレーション切替えベンダリクエストを受信する機能を備えた周辺機器側の制御部は、コンフィギュレーション切替えベンダリクエストの受信時に、バスリセットもしくはUSBケーブル電源断の処理を行うとともに、切り替わった所定のコンフィギュレーションをもつディスクリプタをホスト側に提供し、コンフィギュレーションの内容に従った動作状態へ移行させることを特徴とする周辺機器制御システム。
【請求項４】　請求項３に記載された周辺機器制御システムにおいて、コンフィギュレーション切替え要求に従った動作状態への移行が可能ではない場合に、前記周辺機器側の制御部が拒否応答を返送することを特徴とする周辺機器制御システム。
【請求項５】　請求項１乃至４のいずれかに記載された周辺機器制御システムにおいて、切替えられる前記コンフィギュレーションの１つがデバック メンテナンス通信用のデバイスコンフィギュレーションであることを特徴とする周辺機器制御システム。
【請求項６】　ホストと周辺機器をUSB方式により接続する周辺機器制御システムにおいて、ホストがデータステージ付きのコントロール転送によりポート通信を可能にする送信及び受信のベンダリクエストを発行する機能を備えたデバイスドライバを有することを特徴とする周辺機器制御システム。
【請求項７】　請求項６に記載された周辺機器制御システムにおいて、前記ポート通信がデバック・メンテナンス用の通信に用いられることを特徴とする周辺機器制御システム。
【請求項８】　請求項１乃至７のいずれかに記載された周辺機器制御システムにおいて、前記周辺機器を複合コピー機が装備する機能としたことを特徴とする周辺機器制御システム。
【発明の詳細な説明】
【０００１】
【発明の属する技術分野】本発明は、ホストに複数の周辺機器（例えばプリンタ、スキャナ等）をUSB（Universal Serial Bus）方式により接続し機器を制御するシステムであって、ホストから送信されるベンダデバイスリクエストにより通信路を確保するシステムに関する。
【０００２】
【従来の技術】USB（Universal Serial Bus）は、パーソナルコンピュータ（ＰＣ）および通信（telecom）業界のリーダー達により開発された周辺バス仕様である。USBは、コンピュータ周辺機器のプラグ アンド プレイ機能をＰＣボックスの外部に引き出す。この仕様によって、専用のコンピュータスロットにカードを装着し、周辺機器がＰＣに着脱される度にシステムを再構成する、という必要性をなくす。USBを搭載したパーソナルコンピュータは、コンピュータ周辺機器が物理的に接続された途端に、リブートしたりセットアップを走らせたりする必要なく、当該コンピュータ周辺機器を自動的に構成することができる。USBは、モニタやキーボードのような周辺機器を追加的なプラグインサイトまたはハブとして機能させながら、１台のコンピュータ上で同時に１２７台までの多数のデバイスを走らせることができる。USBの詳細は、１９９６年７月１５日に公表されたユニバーサルシリアルバス仕様、リビジョン1.0および１９９８年８月２３日に公表されたリビジョン1.1に記載されている。この仕様は、コンパック、デジタルイクイップメントコーポレーション、ＩＢＭ　ＰＣカンパニー、インテル、マイクロソフト、ＮＥＣ、およびノーザーンテレコムにより共同で公表されたものである。
【０００３】ここで、USB仕様の内容の概略をUSBバストポロジーにより説明する。USBは、複数のUSBデバイスを１台のUSBホストに接続する。USBデバイスには２つの型（タイプ）、すなわち、ハブ（hub）とファンクション(function)がある。ハブは、USBのための追加的な接続ポイントを提供するデバイスである。ファンクションは、システムに対して、例えば、ＩＳＤＮ接続、デジタルジョイスティック、スピーカ、キーボード、マウス等の機能を提供する。ハブおよびファンクションについて、説明すると、USB物理インターコネクトは、層状スター・トポロジー（tiered star topology）であり、ハブは各スターの中心に位置する。各ワイヤセグメントは、（ａ）ホストとハブとの間、（ｂ）ホストとファンクションとの間、または（ｃ）ハブと他のハブもしくはファンクションとの間、のポイント・ツー・ポイント接続である。図９は、USBシステムのトポロジーを示す。図示のUSBシステムはホスト１０を有する。ホスト１０には２つの接続ポイントであるポート１とポート２がある。ポート２はワイヤセグメント２８によりファンクション２９に接続されている。ポート１はワイヤセグメント１８によりハブ１に接続されている。　ハブ１は５個のポート２０を有する。これらのポートの１つに対して、ワイヤセグメント２４によりファンクション２２が接続されている。同様に、ハブ２とハブ３も、ハブ１のポートに接続されている。ハブ２またはハブ３のポートには種々のファンクション（例えばファンクション３

０、３２等）が接続されている。

【０００４】ハブは、USBのプラグ アンド プレイ アーキテクチャにおけるキー要素である。ハブは、ユーザの立場からのUSB接続を簡略化するよう機能し、低いコストおよび複雑性で、堅牢さ(robustness)を提供する。ハブは、ワイヤリング コンセントレータ（集線装置）であり、USBの多接続特性をもたらす。接続ポイントはポートと呼ぶ（図９参照）。各ハブは、単一の接続ポイントを複数の接続ポイントに変換する。このアーキテクチャは多数のハブの連結をサポートする。一つのハブの他の各下流のポートは、他のハブまたはファンクションへの接続を許容する。ハブは、各下流ポートに対するUSBデバイスの着脱を検出することができ、これらの下流のUSBデバイスに電力を分配することが出来る。各下流ポートは個別にイネーブルされ、フルスピードまたはロースピードのいずれかに構成しうる。ハブは、ロースピードポートを、フルスピード信号(full speed signaling)から切り離す。ファンクションは、データ、または、USBに関する制御情報を送受信するUSBデバイスである。ファンクションは、典型的には、ハブのポートに接続されるケーブルを有する独立した周辺デバイスとして具現化される。しかし、１本のUSBケーブルを有する複数のファンクションと１つの組み込みハブを、１個の物理的パッケージとして具現化してもよい。これは複合デバイスとして知られている。複合デバイスは、ホストにとって、１個または複数個のUSBデバイスが固定的に接続されたハブとして見える。ファンクションの例としては、プリンタ、モニタ、マウス、キーボード、オーディオＣＰＤプレーヤ、テーププレーヤ、ＩＳＤＮ接続等が挙げられる。

【０００５】どのようなUSBシステム上にも１つのホストが存在する。ホストコンピュータシステムに対するUSBインタフェースは、ホストコントローラと呼ばれる。ホストコントローラは、ハードウエア、フィームウエア、またはソフトウエアの組み合わせで具現化される。ルートハブはホストシステムに内蔵され、１個ないし複数個の接続ポイントを提供する。デバイスエンドポイントとは、ホストとUSBデバイスの間の通信フローにおけるソースまたはシンクであるUSBデバイス（ハブまたはファンクション）の一意に識別可能な部分をいう。２つ以上のエンドポイント（EndPoint）を有するUSBファンクションの一例はデータ／ボイスモデムである。これには、ボイスパケット用の１つのエンドポイントと、データパケット用の１つのエンドポイントが存在する。

【０００６】すべてのバストランザクションは、最大３つのパケットの転送に関わる。各トランザクションは、ホストコントローラがスケジュールに従って、トランザクションのタイプおよび方向を示すUSBパケット、USBデバイスアドレスおよびエンドポイント番号を送信するときに、開始される。このパケットはトークンパケット(Token Packet）と呼ばれる。このトークンパケットにより指定されるUSBデバイスは、当該トークンパケットの適当なアドレスフィールドをデコードすることにより、自分自身を選択する。あるトランザクションにおいて、データは、ホストからデバイスへ、またはデバイスからホストへ転送される。データ転送の方法はトークンパケットに指定される。トランザクションのソースは、ついで、その転送が成功したか否かを示すデータパケットを送信する。トランザクションのタイプは４種類ある。１つはコントロール転送でデバイスのコンフィギュレーション（configuration）またはメッセージ転送に使われる。２つめはインタラプト転送でマウス・キーボードなどのイベント通知に使われる。３つめはバルク転送で大量転送を行うために使われる。４つめはアイソクロナス転送でデータの転送の確実性を犠牲にしてデリバリ時間を保証する転送方式である。

【０００７】ホスト以外の最大126個のファンクションを識別するため各ファンクションにアドレスを割り付ける。このアドレスを割り付ける方法はバスエナミュレーションと呼ばれる。ファンクションのアドレスの初期値は０であり、ホストはハブの構成するトポロジに従って各バスのポートを１つずつ有効にしながらアドレス０のデバイスに対してセットアップ動作を行いアドレスを割り付ける。バスエナミュレーション等ファンクションをセットアップするためにホストからファンクションのエンドポイント０に対しコントロール転送によりホストからのリクエストが送られる。このようなファンクションに対するリクエストはデバイスリクエストと呼ばれ、とくにUSBの仕様書では標準デバイスリクエストが定められている。各ファンクションはどのような機能を有するかをディスクリプタ（Descriptor）と呼ばれる形式で保持している。ホストはDescriptorの内容をデバイスリクエストで読みだし、その内容を変更、選択することができる。最終的にSET-CONFIGURATIONデバイスリクエストでファンクションの構成を選択した時点でファンクションのエンドポイント０以外の構成が決定され、それらの機能を使うことができるようになる。

【０００８】ところで、いわゆる複合コピー機(MFP)と呼ばれるプリンタ機能、スキャナ機能、ファックス機能、イメージデータサーバ機能、等各種の機能を外部に公開できる複合機では外部のインタフェース、例えば、パラレル(IEEE1284)、シリアル(RS232C)、Ethernet、USB、IEEE1394などを通じてＰＣなどから各機能へアクセスすることが可能である。とりわけUSBを通信手段とすれば、周辺機器のDescriptorの構成として複数の機器機能のインタフェース（Interface）記述をもつconfigurationを用意することで、ＰＣ側ドライバ／ユーティリティとの対話的な通信が達成できる(以下、この方法を「従来法１」と呼ぶ)。この場合のDescriptor階層構成は、例えば図１０のような階層構成として記述される。

図１０に示す例は、プリンタの例で複数のプリンタインタフェース構成とスキャナポートをInterfaceレベルで持つ。なお、この方法はUSBにおいて多機能デバイスを実現する通常の方法である。また、別の方法としてInterfaceレベルではなくconfigurationレベルでスキャナのあるconfigurationと通常のプリンタのconfigurationをわける方法がある(以下この方法を「従来法２」と呼ぶ)。この場合のDescriptor階層構成は、例えば図１１のような階層構成として記述される。図１１に示す構成では最初のconfigurationが通常のプリンタとしての動作構成で２つめのconfigurationがスキャナ用のものである。このようにすれば前のやり方と違いスキャナ用にEndPointを常時持つ必要は無くなる。

【０００９】一方、周辺機器の開発時デバックや市場でのメンテナンスには従来RS-232C等のシリアルポートやICカードなどが使用されていた。USBをホストとの通信のため備えた装置では、USBを本来の目的のためだけでなく開発ホストやメンテナンス用Note 型ＰＣと通信する方法としても、従来の通信手段に替えて利用することが考えられる。USBを使用することで従来のシリアルなどの通信手段をデバック・メンテナンスのために持つ必要がなくなりコストの削減が可能となる。USBを通信手段として周辺機器内のデバックモニタとの通信やメンテナンスプログラムとの対話的な通信を行う場合、デバック用のインタフェース記述をもつconfigurationが上記従来法１、或いは従来法２と同様に周辺機器のDescriptorの構成として用意される。つまり、図９のスキャナポートに代えてデバックポートを持つか、或いは図１０のスキャナconfigurationに代えてデバック・メンテナンス用configurationを持つようにする。

【００１０】

【発明が解決しようとする課題】しかしながら、従来法１のような構成にした場合、エンドポイントを機能毎に常時用意するため、サポートするエンドポイント数の多い高価なUSBコントローラを使わなければならなくなり、コストアップとなる。また、従来法２ではホスト側がどのconfigurationを選ぶかを制御する必要があり、その方法が難しい。ホスト側のデバイスドライバとして、例えば、プリンタ用とスキャナ用（又はデバック・メンテナンス用）があるときその間でコンフィギュレーションの取り合いが起こることになる。このような状態を調停する仕組みを独自に作り込む必要がある。一方、デバック／メンテナンス用configurationを上記従来法１、或いは従来法２のように持つようにした場合、上記した問題に加え、デバック・メンテナンス用ポートの存在がホストのユーザに知られてしまうという問題もある。

【００１１】本発明は、上記した従来技術の問題点に鑑みてなされたもので、その目的は、従来法１のように多くのエンドポイントを必要とせず、従来法２のようにホスト側デバイスドライバを複雑化せず、複数のデバイス用のUSBの通信路が確保でき、さらに、公開されたDescriptorからデバック・メンテナンス用ポートの存在を知られることがないようにした周辺機器制御システムを提供することにある。具体的には、ベンダリクエストとしてデバイスDescriptor切替え要求を用意し、この要求の後、ホスト側からバスリセットをかけセットアップをやり直すことで別のデバイスのconfigurationを起こさせるようにした手段を設けたシステムを提供することにある。また、送信及び受信ベンダリクエストを用いて、データステージ付のコントロール転送を行うことによりポート通信を可能にするファンクション用デバイスドライバをホスト側に分離して用意したシステムを提供することにある。

【００１２】

【課題を解決するための手段】請求項１の発明は、ホストと周辺機器をUSB方式により接続する周辺機器制御システムにおいて、ホストがコンフィギュレーション切替えベンダリクエストを発行する機能を備えたデバイスドライバを有することを特徴とする周辺機器制御システムである。

【００１３】請求項２の発明は、請求項１に記載された周辺機器制御システムにおいて、前記ホストは、デバイスドライバをコンフィギュレーション毎に有することを特徴とするものである。

【００１４】請求項３の発明は、請求項１又は２に記載された周辺機器制御システムにおいて、前記コンフィギュレーション切替えベンダリクエストを受信する機能を備えた周辺機器側の制御部は、コンフィギュレーション切替えベンダリクエストの受信時に、バスリセットもしくはUSBケーブル電源断の処理を行うとともに、切り替わった所定のコンフィギュレーションをもつディスクリプタをホスト側に提供し、コンフィギュレーションの内容に従った動作状態へ移行させることを特徴とするものである。

【００１５】請求項４の発明は、請求項３に記載された周辺機器制御システムにおいて、コンフィギュレーション切替え要求に従った動作状態への移行が可能ではない場合に、前記周辺機器側の制御部が拒否応答を返送することを特徴とするものである。

【００１６】請求項５の発明は、請求項１乃至４のいずれかに記載された周辺機器制御システムにおいて、切替えられる前記コンフィギュレーションの１つがデバック メンテナンス通信用のデバイスコンフィギュレーションであることを特徴とするものである。

【００１７】請求項６の発明は、ホストと周辺機器をUSB方式により接続する周辺機器制御システムにおいて、ホストがデータステージ付きのコントロール転送によりポート通信を可能にする送信及び受信のベンダリクエストを発行する機能を備えたデバイスドライバを有するこ

とを特徴とするものである。

【００１８】請求項７の発明は、請求項６に記載された周辺機器制御システムにおいて、前記ポート通信がデバック・メンテナンス用の通信に用いられることを特徴とするものである。

【００１９】請求項８の発明は、請求項１乃至７のいずれかに記載された周辺機器制御システムにおいて、前記周辺機器を複合コピー機（コピー機能、プリンタ機能、スキャナ機能、ファックス機能、イメージデータサーバ機能、等各種の機能を装備するコピー機）が装備する機能としたことを特徴とするものである。

【００２０】

【発明の実施の形態】本発明を添付する図面とともに示す以下の実施例に基づき説明する。図１は、本発明に係わる周辺機器制御システムの構成の概要を示すブロック図を示す。なお、以下の説明ではUSBファンクション（周辺機器）の例としてプリンタを装備した複合機を、USBホストとしてWindows98を搭載したPCを想定して説明を行う。USBホストのソフトウエアとハードウエアの構成は、USB仕様書リビション1.1の第１０章に規定されている。図１に示すように、USBホスト１００のハードウエア９０は、SIE(Serial Interface Engine)９２とUSBホストコントローラ９１からなる。SIE９２はUSBの仕様書で規定されている差動信号を解釈する部分でありUSBバス(途中にHubがあってもよい)を介してUSBファンクション２００と接続されている。USBホスト１００のUSBハードウエア９０は階層構造をもつUSBドライバ７０から操作される。USBドライバ７０を介してアプリケーション５０はUSBファンクション２００と通信を行い機能を達成する。USBファンクション２００は、例えばプリンタを装備とした複合機等の周辺機器であり、USBの通信を行うUSBハードウエア８０はSIE８２とUSBファンクションコントローラ８１からなる。USBハードウエア８０はUSBドライバ６０により管理されており、デバイス機能４０よりアクセスされる。デバイス機能４０は、例えばプリンタ機能の場合、プリント動作及びプリンタ情報管理を行う部分であり、USBドライバ６０を介してUSBホスト１００側からのプリント要求等の動作指示やデータを送受信し所望の動作を行う。

【００２１】図２はUSBホスト１００側のUSBドライバ層の構成を示したものである。ホスト側USBドライバ７０は階層構造を持つかたちで実装される。例えばWindows98ではWDMとよばれるドライバの階層構造が規定されている。USBハードウエア９０はUSBドライバ７０を構成するドライバ層のUSBバスクラスドライバ７１によって管理される。USBバスクラスドライバ７１はできるだけハードウエアに依存しないかたちで作られておりハードウエア依存部分はミニドライバ(MD)という部分で吸収する構造をとっている(WDMの場合)。この例ではOHCIタイプのハードウエアを管理するOHCI MD７６ｍを持っている。USBに接続された各周辺機器の機能を利用するためのドライバはクラスドライバと呼ばれる。クラスドライバは機器の機能をバスにもできるだけ依存しないかたちで実装されている。例えばUSBバスクラスドライバが1394バスクラスドライバになったとしてもできるだけ影響を受けないかたちで作られており、その依存部分もやはりミニドライバ（MD）が吸収するように構成される。図２の例では、デバイス機能を利用するためのドライバであるStreamingクラスドライバ７２、HIDクラスドライバ７３、プリンタクラスドライバ７４、scannerクラスドライバ７７は、それぞれのMDとしてスピーカMD７２ｍ、マウスMD７３ｍ、プリンタMD７４ｍ、scannerMD７７ｍを備える。

【００２２】次に、USBホストが送信するデバイスリクエスト（ベンダデバイスリクエスト）によるUSBファンクションへの接続制御動作についてその実施例を説明する。先ず、第１の接続制御方式について説明する。第１の接続制御方式は、ベンダデバイスリクエストとしてデバイスディスクリプタ切替え要求を用意し、この要求の後、ホスト側からバスリセットをかけセットアップをやり直すことで別のコンフィギュレーションによるデバイスのコンフィギュレーションを起こさせる方式である。この方式によると、ホスト側では複雑な制御の必要が無く、各機能毎のデバイスドライバを用意するだけで、プリンタ機能、スキャナ機能、イメージデータサーバ機能といった機能をこれら全てに必要なエンドポイントをもつ事無くこの制御が実現可能である。図２に示す実施例では、プリンタクラスドライバとscannerクラスドライバがエンドポイントを共通にし、切り替え要求に対応するクラスドライバであり、それぞれは全く独自に作成されるが、ただ、ベンダリクエストとして“デバイスディスクリプタ切替え要求”を送信する機能をもたせるだけで済む。

【００２３】図３は、第１の接続制御方式で用いるベンダデバイスリクエストを示す。ベンダデバイスリクエストは、図３に示すように、５つのフィールドからなる（これはUSB仕様書に標準化されている）。最初のbmRequestTypeでベンダリクエストであることを示し、次のbRequestで特定のリクエストを識別する。この例の場合、0x71を“デバイスディスクリプタ切替え要求”としている。図２に示すホストのUSBドライバ７０が備えるプリンタクラスドライバ７４とスキャナクラスドライバ７７はこのリクエストを送出する機能を有する。このベンダリクエストを送出する機能以外に特殊な処理は必要ない。また、それぞれのドライバは全く独立に動作し相互の関係はない。

【００２４】図５は、USBファンクション２００のデバイスドライバ６０の構成を示したものである。デバイスドライバ６０はUSB Function Controller(以下「UFC」と記す)８１のレジスタを介して操作を行う。通常UFC8

１内部ではエンドポイント(以下EPと略す)ごとのデータ送受信をおこなうように構成されており、この例ではEP1とEP2はDMAによりバルク転送を行うことができる。EP0のデータはレジスタアクセスでありEP0により標準デバイスリクエスト、クラスデバイスリクエストおよびベンダデバイスリクエストを受け付けて処理をおこなう。それぞれのEPはマネージャ層により管理される。これらはUFC８１からの割り込みによって動作する割り込みハンドラである。EP0マネージャ６４は、EP0のコントロール転送を解釈する。とくにデバイスリクエストで必要とされるディスクリプタの管理は、ディスクリプタ管理部６２で行われる。EP1マネージャ６４は、バルクアウト転送により受信動作を管理し受信データ管理部６６にデータを渡す。EP2マネージャ６５は、バルクイン転送によるデータ転送を管理し送信データ管理部６７にあるデータを転送する。これらの部分は、USBデバイスI/F６１により抽象化されたI/FでUSBプリントアプリ４１、USB scannerアプリ４３と交信する。

【００２５】EP0マネージャ６３はコントロール転送およびUSBケーブルの電源ラインの監視をおこなう。電源ラインの監視は、活線挿抜(プラグ アンド プレイ)を監視するためのもので、USBケーブルのVcc入力電圧があるか否かでケーブルの抜き差しを検出する。EP0マネージャ６３の管理するコントロール転送には標準デバイスリクエスト、クラスデバイスリクエスト、ベンダデバイスリクエストがある。図６はEP0マネージャの動作の一部であるデバイスリクエスト処理を示すフローチャートである。図６に示す処理フローでは、ベンダデバイスリクエスト（図３）におけるbmRequestTypeに指示されるデバイスリクエストのタイプ、即ち標準デバイスリクエスト、クラスデバイスリクエスト、ベンダデバイスリクエストがそれぞれ判別され（Ｓ６０１，Ｓ６０７，Ｓ６０９）、各デバイスリクエストごとに処理を分けて行う。

【００２６】図６のフローに従いデバイスリクエスト処理動作を説明すると、まず、標準デバイスリクエストであるかがチェックされ（Ｓ６０１）、標準デバイスリクエストである場合、そのリクエスト内容がGET-Descriptorリクエストであるかがチェックされ（Ｓ６０２）、GET-Descriptorリクエストである場合には、ディスクリプタ管理部６２からリクエストされたDescriptorのデータを取得して（Ｓ６０３）、得たデータをデータステージとしてUSBホスト１００に転送し（Ｓ６０４）、その後、ステータスステージ処理を行う（Ｓ６０５）。この時、ディスクリプタ管理部６２は「第１の接続制御方式」の処理のため２つの全く異なったデバイス（本例ではプリンタとスキャナ）のためのテーブルを管理しており、それを切替えることができるようになっている。切替え後に選択されているデバイスのデータは、上記のように、GET-Descriptorリクエストの処理で要求されたディスクリプタのデータとしてUSBホスト１００に転送される。

【００２７】デバイスリクエストのbmRequestTypeがベンダデバイスリクエストである場合（Ｓ６０９）、さらにbRequest（図３参照）に指示されるベンダリクエストの種類を識別する。識別した結果、ベンダリクエストが0x71、即ちディスクリプタ切替え要求で有る場合に（Ｓ６１０）、まずUSBデバイスI/F６１に対して切替え要求があったことを通知する（Ｓ６１１）。USBデバイスI/F６１では、切り替え可能な状態であるかをチェックし（Ｓ６１２）、可能であれば、利用しているデバイスの機能、つまり、アクティブ化するアプリをUSBプリントアプリ４１、USB scannerアプリ４３の一方から他方へ切り替える（Ｓ６１３）。次いで、ディスクリプタ管理部６２が管理するディスクリプタの切替え処理を行う（Ｓ６１３）。処理がうまくいった場合にステータスステージでそれをステータスACKとしてUSBホスト１００に返送する（Ｓ６１４）。

【００２８】この後に、コネクト信号断処理を行う（Ｓ６１５）。これはハードウエアで差動信号のプルアップ抵抗を操作するためのもので、本実施例はバルク転送可能なフルスピードデバイスであるためD+信号にプルアップ抵抗が挿入されており、これを切断する処理を実行する。本実施例ではこの抵抗ラインを一旦切ることでケーブルを引き抜いた状態をつくり、その後プルアップを復活させることでケーブルを挿入した状態とおなじ効果を得る。これにより、USBホスト１００側では、いままでつながっていたデバイスが無くなったと解釈し、USBバスドライバ７０はそれをクラスドライバに通知する。また、その後、プルアップを復活させることでUSBホスト１００側では新しいデバイスが接続されたと解釈し、デバイスのディスクリプタ取得動作がUSBバスドライバ７０により行われる。このとき切り替わったディスクリプタが取得されるため、それに従ったクラスドライバがアクティブにされる。つまり、ディスクリプタ切替え要求前にプリンタとしてのディスクリプタを提供していたとすると電源断/復活の処理でホスト側はプリンタクラスドライバ７４を無効にして、新たに与えられたscannerポートのディスクリプタに従ってscannerクラスドライバ７７を有効にする。なお、コネクト信号断処理をこの実施例ではファンクション側のハードウエアで対応したがUSBホスト１００側のUSBバスクラスドライバ７１を操作することで擬似的に起こす方法もある。

【００２９】Ｓ６１２で切り替え可能な状態であるかをチェックした結果、アプリが動作中等であり、切り替え可能な状態ではない場合、デバイスの切替えを禁止する必要がある。例えば、スキャナのデータ転送中はプリンタへの切替えを禁止する。切替えを拒否する場合、ベンダリクエストのデータステージでNAKをUSBホスト１００に送出し（Ｓ６１６）、この処理を終える。NAKは通常そのリクエストが受け入れ可能ではない事を示す。切替

え要求を発行したホスト側デバイスドライバ、アプリケーションは、NAKが送信されてくることで、切替えが不可能であることを知り、ダイアログの表示等対応した動作を行う。

【００３０】上記の例ではプリンタとスキャナの機能の切替えを示したが、他にイメージサーバ機能(MFP側で印刷を行った画像データをハードディスク等に記憶しておきそれを再利用するための機能)、ファックス機能等、MFPの持つ機能を切替える事ができるようにする事が可能である。ユーザ毎にこれらの機能を利用するかどうかの利用頻度はかなり異なる。つまり、すべての機能を常に利用可能にする必要はなく一度に全ての機能を利用できなくてもあまり使用者の不便とはならない。切替え機能を実現する事によりこれらのデバイス機能の全てに付いてエンドポイントを用意する事無く安価なUSBコントローラを用い外部からこれらの機能を利用する事ができる。

【００３１】さらに、上記したベンダデバイスリクエストによるUSBファンクションへの接続制御を行う場合、切替えるデバイス機能の１つをデバック・メンテナンスPort（以下「Debug Port」と記す）機能とした実施例について以下に説明する。図７は、USBホスト１００側のUSBドライバ層の構成を示したものである。USBドライバ層は、図７に示すように、ドライバの階層構造の１つとして、Debug Portクラスドライバ７５を備え、Debug Portクラスドライバ７５とUSBバスクラスドライバ７１との間にDebug PortMD７５ｍを備えており、それ以外の点で、図２と基本的な相違はない。また、図８は、USBファンクション２００側のデバイスドライバ６０の構成を示したものである。これは、図７のUSBホスト１００側のUSBドライバに対応したもので、図８に示すように、USBDebug Portアプリ４２が、USBドライバによって制御されるデバイスの１つとして装備される。この点以外に図８のUSBファンクション側のデバイスドライバは、図５に示すドライバと基本的に相違しない。接続の制御動作の手順は、図６を参照して説明した先のscannerポートのデバイスドライバとの接続手順と変わりがないので重複する説明はしない。このように、ディスクリプタ切替え要求ベンダリクエストを送信する機能をホスト側デバイスドライバが備え、ファンクション側ではこれを認識しディスクリプタの切替え機能、ドライバを利用するデバイス機能に対する通知機能、コネクト断処理をもつこの接続制御システムによれば、ホスト側、ファンクション側とも別の機能をもつデバイスに切り替わって処理することができ、とくに、その機能がデバッグメンテナンス用の動作である場合には、これによってディスクリプタとしてデバックポートの機能を公開せず、デバッグポート用にエンドポイントを用意する事無くデバッグ・メンテナンス動作を行うことが可能となる。

【００３２】次に第２の接続制御方式について説明する。第２の接続制御方式は、データの転送にデータステージ付のコントロール転送を行うことによるものである。２つのベンダリクエストを送信することにより特定のファンクションとの接続を行い、別のポート通信を可能とする。ホスト側のファンクション用デバイスドライバには、通信のための処理を特定のファンクション用ドライバとして分離して用意する。図７に示すプリンタクラスドライバ７４とDebug Portクラスドライバ７５を備えるクラスドライバの例で説明すると、ホスト側にDebug Portクラスドライバ７５が用意される。ここでは、さらに機器本来のデバイスドライバであるプリンタクラスドライバ７４にDebug Port通信のための送信・受信のベンダリクエストの処理を委譲している。即ち、Debug Portクラスドライバ７５のDebug Port MD７５ｍからの要求をプリンタクラスドライバ７４により受け持つように、プリンタMD７４ｍを設け、ここからデバッグ送受信リクエストを発行する。ファンクション側ではこのベンダリクエストに対する処理をデバイス機能として持つ。これによりディスクリプタとしてデバックポートの機能を公開せずにデバッグポート用にエンドポイントを用意する事無くデバッグ・メンテナンス動作を行うことができる。

【００３３】図４は、第２の接続制御方式で用いられるベンダデバイスリクエストを示す。ベンダデバイスリクエストは、図４に示すように、５つのフィールドからなる（USB仕様書で標準化）。最初のbmRequestTypeでベンダリクエストであることを示し、次のbRequestで特定のリクエストを識別する。この例ではbRequestが0x72のものがデバック・メンテナンスデータ送信用リクエスト（図４（Ａ））であり、0x73のものがデータ受信用リクエスト（図４（Ｂ））である。wLengthでデータの長さが指定されリクエストのデータステージで実際のデータがやりとりされる。プリンタMD７４ｍはDebug Port MD７５ｍから要求を受けるとこの２つのベンダリクエストを送出する。また、リクエストの結果等として返却されてくるデータはプリンタMD７４ｍからDebug Port MD７５ｍに渡される。また、デバイスの状態問い合わせ処理などもDebug Port MD７５ｍからプリンタMD７４ｍに処理が委譲される処理がなされる。

【００３４】ベンダリクエスト0x72/0x73に対する処理は、USBファンクション２００側のデバイスドライバのEP0マネージャ６３（図８参照）の動作の一部であるデバイスリクエスト処理により行われる。これは、先に示した図６に示したフローチャートに従って実行される。図６に示す処理フローでは、ベンダデバイスリクエスト（図４参照）におけるbmRequestTypeに指示されるデバイスリクエストのタイプ、即ち標準デバイスリクエスト、クラスデバイスリクエスト、ベンダデバイスリクエストがそれぞれ判別され（Ｓ６０１，Ｓ６０７，Ｓ６０９）、各デバイスリクエストごとに処理を分けて行う。

bmRequestTypeのデバイスリクエストが標準デバイスリクエスト或いはクラスデバイスリクエストではなく（Ｓ６０１、Ｓ６０７）、ベンダリクエストであると判断される場合（Ｓ６０９）、さらにbRequest（図４参照）に指示されるベンダリクエストの種類を識別する（Ｓ６１０，Ｓ６１７，Ｓ６２０）。

【００３５】識別した結果、ベンダリクエストが0x72、即ち、デバック・メンテナンスデータ送信用リクエストで有る場合に（Ｓ６１７）、0x72ベンダリクエストは受信(ホスト側からは送信)のデータステージをもつ（Ｓ６１８）。この処理ではベンダリクエスト（図４（Ａ）参照）中にあるデータ長(wLength)に従った長さのデータを受信する。受信されたデータはUSBデバイスI/F６１を介してUSBデバッグポートアプリ４２が受け取る。又、受信に成功したかどうかをステータスステージでホスト側に返送する（Ｓ６１９）。ベンダリクエストが0x73、即ち、デバック・メンテナンスデータ受信用リクエストで有る場合に（Ｓ６２０）、0x73ベンダリクエストは送信(ホスト側からは受信)のデータステージをもつ（Ｓ６２１）。ベンダリクエスト（図４（Ｂ）参照）中にあるデータ長に従ってデータステージではUSBデバイスI/F６１を介してUSBデバックポートアプリ４２から与えられるデータを転送する。そののちホスト側から転送に成功した場合のステータス情報が返送される（Ｓ６２２）。このように、プリンタクラスドライバを介して行われる２つのベンダリクエストによりホスト側デバックポートドライバとファンクション側デバックポートアプリとの通信が可能となる。

【００３６】

【発明の効果】（１）　請求項１の発明に対応する効果

ホストがコンフィギュレーション切替えベンダリクエストを発行する機能を備えたデバイスドライバを有することにより、複数のデバイス（周辺機器）の機能を利用するにあたり、これらの機能全てに必要なエンドポイント数をもつこと無く、少ない数のエンドポイントで通信路を確保することができる。

（２）　請求項２の発明に対応する効果

上記（１）の効果に加えて、ホスト側のデバイスドライバをコンフィギュレーション毎に用意することにより、複雑な制御を必要とすることのない単純な構成のデバイスドライバを用いることができる。

（３）　請求項３の発明に対応する効果

周辺機器の制御部がコンフィギュレーション切替えベンダリクエストの受信時に、バスリセットもしくはUSBケーブル電源断の処理を行い、切り替え後のコンフィギュレーションをもつディスクリプタをホスト側に提供し、このコンフィギュレーションに従った動作状態へ移行させる動作を行うようにしたことにより、請求項１，２の発明を容易に実施することができる。

（４）　請求項４の発明に対応する効果

上記（３）の効果に加えて、切替え要求を発行したホスト側デバイスドライバ、アプリケーションは、デバイス側からNAKを受け取り切替えが不可能であることを知ることにより、ダイアログの表示等対応した動作を行うようにすることが可能となり、ユーザにとり利便性が向上する。

（５）　請求項５の発明に対応する効果

上記（１）～（４）の効果に加えて、切替えられる前記コンフィギュレーションの１つをデバック メンテナンス通信用のデバイスコンフィギュレーションとすることにより、ディスクリプタとしてデバックポートの機能を公開せずにすみ、又デバッグポート用にエンドポイントを用意すること無く、デバッグ・メンテナンス動作を行うことができる。

【００３７】（６）　請求項６の発明に対応する効果

ホストがデータステージ付きのコントロール転送によりポート通信を可能にする送信及び受信のベンダリクエストを発行する機能を備えたデバイスドライバを有することにより、複数のデバイス（周辺機器）の機能をこれらの全てに必要なエンドポイント数をもつこと無く、少ない数のエンドポイントでこれらの機能を利用するための通信路を確保することができる。また、送信及び受信のベンダリクエストを発行する機能を用いたホスト側のデバイスドライバが機器本来のデバイスドライバへこのベンダリクエストの処理を委譲して受け持たせ、機器側ではこのベンダリクエストに対する処理をデバイス機能として持つようにすることにより、ディスクリプタとしてこのポートの機能を公開せず、このポート通信用にエンドポイントを用意すること無く、デバイスを動作させることが可能となる。

（７）　請求項７の発明に対応する効果

上記（６）の効果に加えて、本発明に係わるポート通信をデバック・メンテナンス用の通信に用いることにより、ディスクリプタとしてデバックポートの機能を公開せずにすみ、又デバッグポート用にエンドポイントを用意すること無く、デバッグ・メンテナンス動作を行うことができる。

（８）　請求項８の発明に対応する効果

周辺機器を複合コピー機（例えば複写機能、プリンタ機能、ファクシミリ機能、スキャナ機能、電子ファイリング機能等を複合して持つ機械）が装備する各機能とし、周辺機器制御システムを構成することにより、複合コピー機を要素とする周辺機器制御システムにおいて上記（１）～（７）効果を実現することにより、システムの性能を向上させることができる。

【図面の簡単な説明】

【図１】　本発明に係わる周辺機器制御システムの構成の概要を示すブロック図を示す。

【図２】　USBホストのドライバ層の１構成例を示す。

【図３】　“デバイスディスクリプタ切替え要求”に用

いるベンダデバイスリクエストを示す。

【図４】　データステージ付のコントロール転送による送受信に用いるベンダリクエストを示す。

【図５】　USBファンクションのデバイスドライバの１構成例を示す。

【図６】　EP0マネージャの動作の一部であるデバイスリクエスト処理のフローチャートを示す。

【図７】　USBホストのドライバ層の他の構成例を示す。

【図８】　USBファンクションのデバイスドライバの他の構成例を示す。

【図９】　USBシステムのトポロジーを示す。

【図１０】　従来の周辺機器のDescriptorの階層構成の一例を示す。

【図１１】　従来の周辺機器のDescriptorの階層構成の他の例を示す。

【符号の説明】

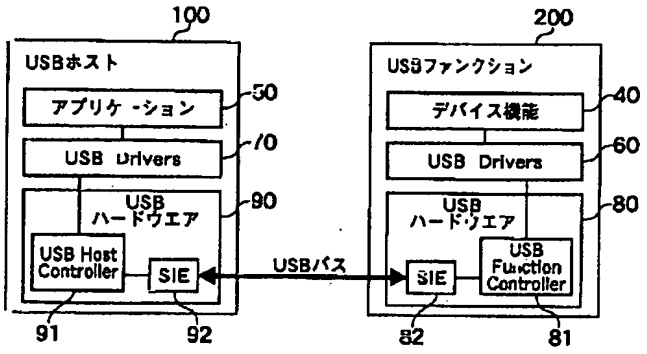
４０…デバイス機能、　４１…USBプリントアプリ、４２…USBDebug Port アプリ、　４３…USBscannerアプリ

５０…アプリケーション、　６０…USBドライバ（ファンクション側）、６１…USBデバイスI/F、

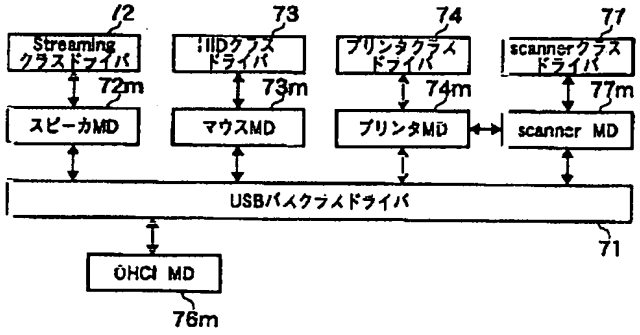
６２…ディスクリプタ管理部、６３…EP0マネージャ、　７０…USBドライバ（ホスト側）、７１…USBバスクラスドライバ、　７４…プリンタクラスドライバ、７５…Debug Port クラスドライバ、　７７…scannerクラスドライバ、１００…USBホスト、

２００…USBファンクション。

【図１】



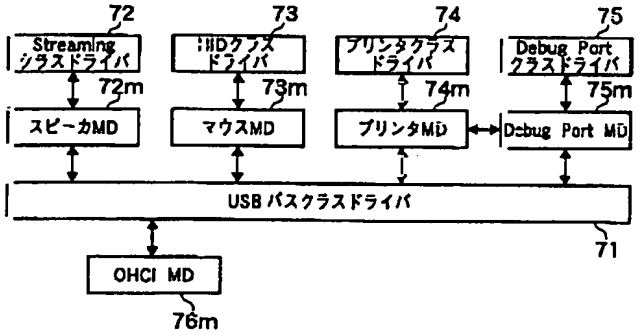
【図２】



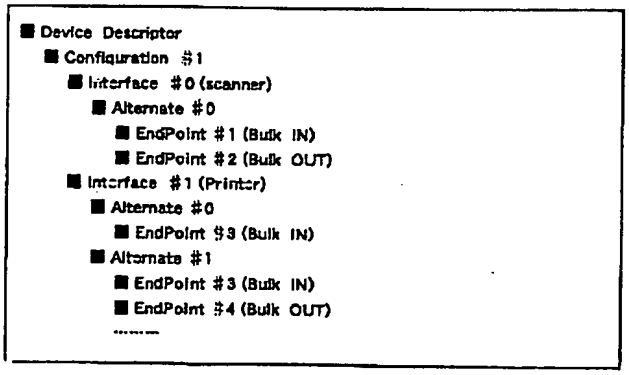
【図３】

bmRequestType	0x40	(ベンダリクエスト)
bRequest	0x71	(特定のリクエストを示す)
wValue	0x0000	(bRequest 引数)
wIndex	0x0000	(bRequest 依存の indexまたは offset 値)
wLength	0x0000	データステージバイト数

【図７】



【図１０】



【図4】

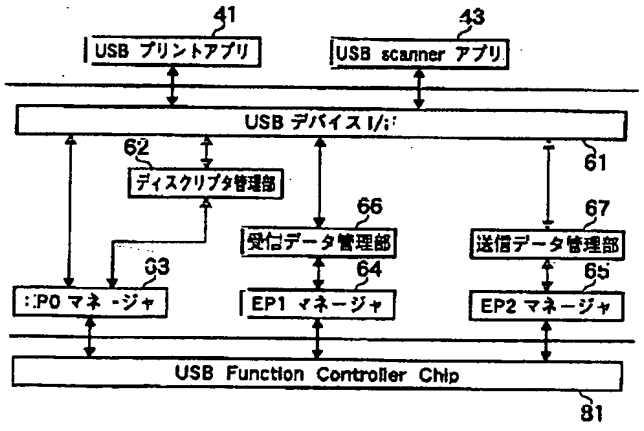
(A)

bmRequestType	0x40	(ベンダリクエスト/データ送信)
bRequest	0x72	(特定のリクエストを示す)
wValue	0x0000	(bRequest 引数)
wIndex	0x0000	(bRequest 依存の indexまたは offset 値)
wLength	0x000a	データステージバイト数

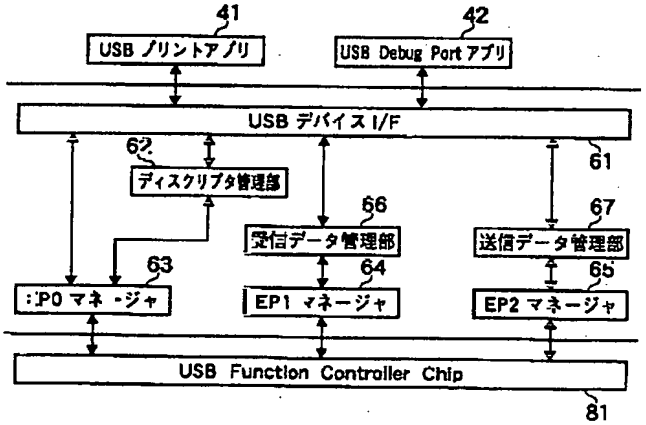
(B)

bmRequestType	0xd0	(ベンダリクエスト/データ受信)
bRequest	0x73	(特定のリクエストを示す)
wValue	0x0000	(bRequest 引数)
wIndex	0x0000	(bRequest 依存の indexまたは offset 値)
wLength	0x000a	データステージバイト数

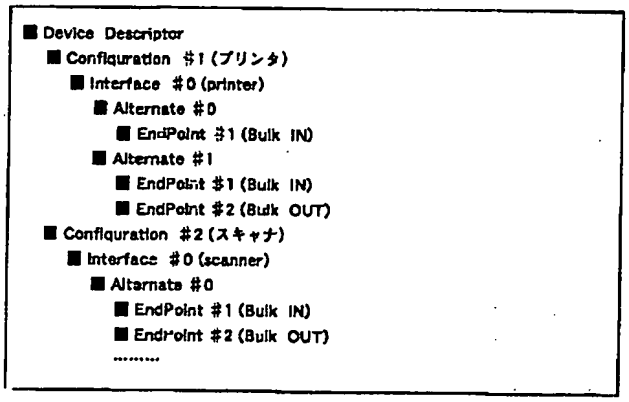
【図5】



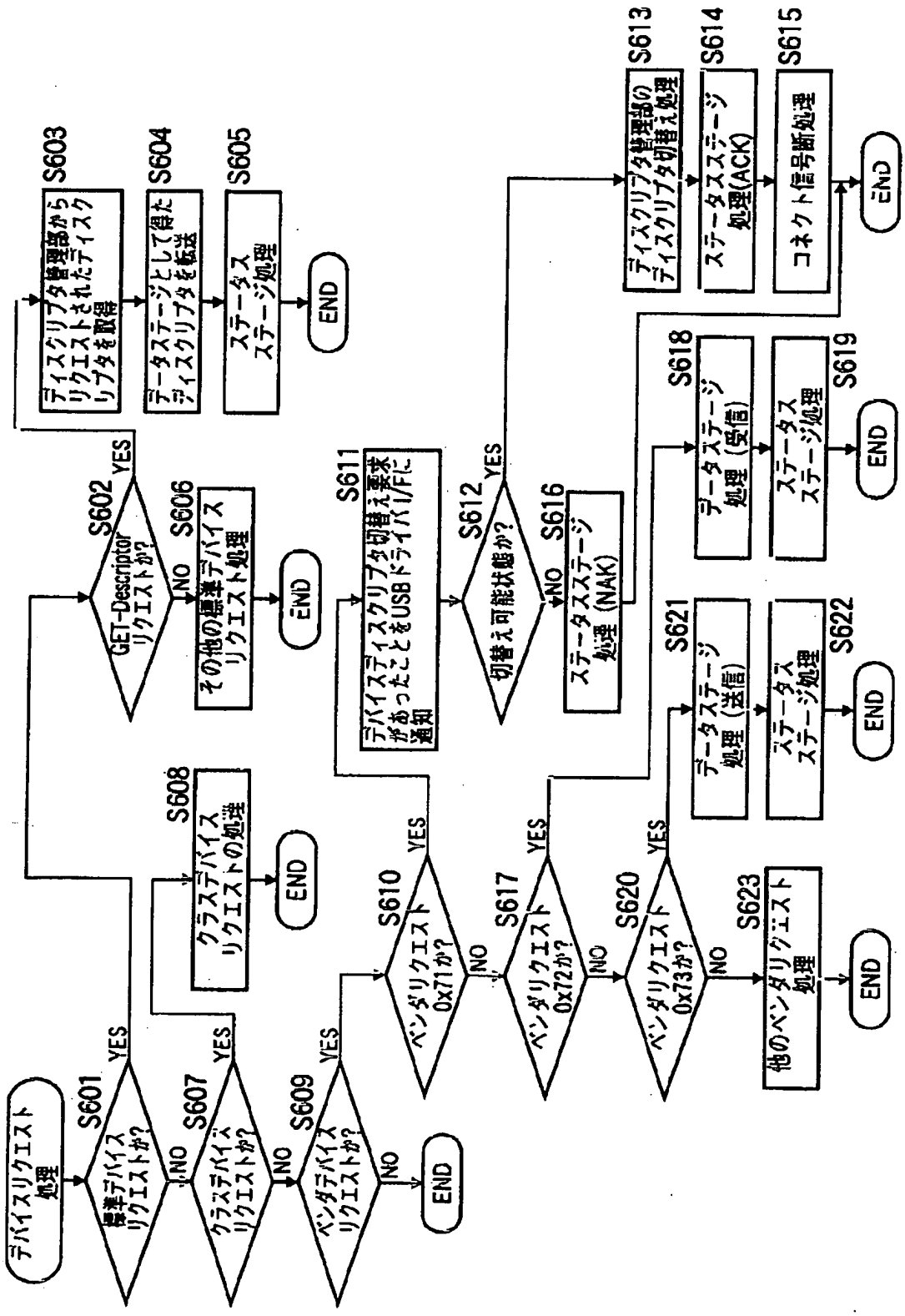
【図8】



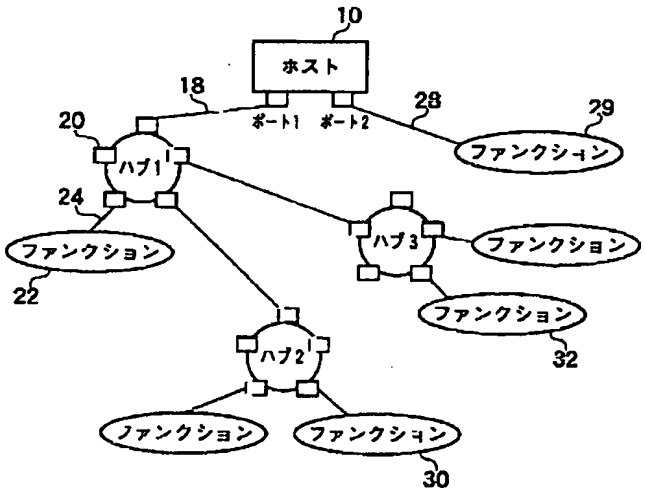
【図11】



【図6】


デバイスリクエスト処理
S601
標準デバイスリクエストか?
YES
NO
S602
GET-Descriptorリクエストか?
YES
NO
S603
ディスクリプタ管理部からリクエストされたディスクリプタを取得
S604
データステージとして得たディスクリプタを転送
S605
ステータスステージ処理
END
S606
その他の標準デバイスリクエスト処理
END
S607
クラスデバイスリクエストか?
YES
NO
S608
クラスデバイスリクエストの処理
END
S609
ベンダデバイスリクエストか?
YES
NO
END
S610
ベンダリクエスト0x71か?
YES
NO
S611
デバイスディスクリプタ切替え要求があったことをUSBドライバI/Fに通知
S612
切替え可能状態か?
YES
NO
S613
ディスクリプタ管理部のディスクリプタ切替え処理
S614
ステータスステージ処理(ACK)
S615
コネクト信号断処理
END
S616
ステータスステージ処理(NAK)
S617
ベンダリクエスト0x72か?
YES
NO
S618
データステージ処理(受信)
S619
ステータスステージ処理
END
S620
ベンダリクエスト0x73か?
YES
NO
S621
データステージ処理(送信)
S622
ステータスステージ処理
END
S623
他のベンダリクエスト処理
END

【図9】


10
ホスト
18
ポート1
ポート2
28
29
ファンクション
20
ハブ1
24
ファンクション
22
ハブ3
ファンクション
ファンクション
32
ハブ2
ファンクション
ファンクション
30